pyppmd

Hiroshi Miura

Nov 15, 2022

CONTENTS:

PPM, Prediction by partial matching, is a wellknown compression technique based on context modeling and prediction. PPM models use a set of previous symbols in the uncompressed symbol stream to predict the next symbol in the stream.

PPMd is an implementation of PPMII by Dmitry Shkarin.

The `pyppmd` package uses core C files from `p7zip`. The library has a bare function and no metadata/header handling functions. This means you should know compression parameters and input/output data sizes.

This library implements PPMd Variant H, and PPMd Variant I Version 2.

CHAPTER

ONE

GETTING STARTED

1.1 Install

The pyppmd is written by Python and C language bound with both CFFI and CPython C/C++ API, and can be downloaded from PyPI(aka. Python Package Index) using standard 'pip' command as like follows;

```
pip install pyppmd
```

When installing on CPython, it downloads a wheel with CPython C/C++ extension. When installing on PyPY, it downloads a wheel with CFFI extension. There are binaries for CPython 3.6, 3.7, 3.8, 3.9 on Windows(32bit, 64bit), macOS and Linux(amd64, aarch64), and PyPy7.3(python 3.7) on macOS, Windows(64bit), and Linux(32bit, aarch64).

CHAPTER

TWO

APPLICATION PROGRAMMING INTERFACE

2.1 Exception

exception PpmdError

This exception is raised when an error occurs.

2.2 Simple compression/decompression

This section contains:

- function *compress()*
- function *decompress()*

compress(*bytes_or_str: Union[bytes, bytearray, memoryview, str]*, *max_order: int*, *mem_size: int*, *variant: str*)

Compress *bytes_or_str*, return the compressed data.

Parameters

- **bytes_or_str** (*bytes-like object or str*) – Data to be compressed. When it is type of str, encoded with "UTF-8" encoding before compress.
- **max_order** (*int*) – maximum order of PPMd algorithm
- **mem_size** (*int*) – memory size used for building PPMd model
- **variant** (*str*) – PPMd variant name, only accept "H" or "I"

Returns

Compressed data

Return type

bytes

```
compressed_data = compress(data)
```

decompress_str(*data: Union[bytes, memoryview]*, *max_order: int*, *mem_size: int*, *encoding: str*, *variant: str*)

Decompress *data*, return the decompressed text.

When encoding specified, return the decoded data as str type by specified encoding. Otherwise it returns data decoding by default "UTF-8".

Parameters

- **data** (*bytes-like object*) – Data to be decompressed.

- **max_order** (*int*) – maximum order of PPMd algorithm
- **mem_size** (*int*) – memory size used for building PPMd model
- **encoding** (*str*) – Encoding name to use when decoding raw decompressed data
- **variant** (*str*) – PPMd variant name, only accept “H” or “I”

Returns
Decompressed text

Return type
str

Raises
PpmdError – If decompression fails.

```
decompressed_text = decompress_str(data)
```

decompress(*data: Union[bytes, memoryview]*, *max_order: int*, *mem_size: int*, *variant: str*)

Decompress *data*, return the decompressed data.

Parameters

- **data** (*bytes-like object*) – Data to be decompressed
- **max_order** (*int*) – maximum order of PPMd algorithm
- **mem_size** (*int*) – memory size used for building PPMd model
- **variant** (*str*) – PPMd variant name, only accept “H” or “I”

Returns
Decompressed data

Return type
bytes

Raises
PpmdError – If decompression fails.

```
decompressed_data = decompress(data)
```

2.3 Streaming compression

class PpmdCompressor

A streaming compressor. It’s thread-safe at method level.

__init__(*self*, *max_order: int*, *mem_size: int*, *variant: str*, *restore_method: int*)

Initialize a PpmdCompressor object. restore_method param is affected only when variant is “I”.

Parameters

- **max_order** (*int*) – maximum order of PPMd algorithm
- **mem_size** (*int*) – memory size used for building PPMd model
- **variant** (*str*) – PPMd variant name, only accept “H” or “I”
- **restore_method** (*int*) – PPMD8_RESTORE_METHOD_RESTART(0) or PPMD8_RESTORE_METHOD_CUTOFF(1)

compress(*self*, *data*)

Provide data to the compressor object.

Parameters
data (*bytes-like object*) – Data to be compressed.

Returns
A chunk of compressed data if possible, or `b''` otherwise.

Return type
bytes

flush(*self*)

Flush any remaining data in internal buffer.

The compressor object can not be used after this method is called.

Returns
Flushed data.

Return type
bytes

```
c = PpmdCompressor()

dat1 = c.compress(b'123456')
dat2 = c.compress(b'abcdef')
dat3 = c.flush()
```

2.4 Streaming decompression

class PpmdDecompressor

A streaming decompressor. Thread-safe at method level. A restore_method param is affected only when variant is "I".

__init__(*self*, *max_order: int*, *mem_size: int*, *variant: str*, *restore_method: int*)

Initialize a PpmdDecompressor object.

Parameters

- **max_order** (*int*) – maximum order of PPMd algorithm
- **mem_size** (*int*) – memory size used for building PPMd model
- **variant** (*str*) – PPMd variant name, only accept "H" or "I"
- **restore_method** (*int*) – PPMD8_RESTORE_METHOD_RESTART(0) or PPMD8_RESTORE_METHOD_CUTOFF(1)

decompress(*self*, *data*, *max_length=-1*)

Decompress *data*, returning decompressed data as a `bytes` object.

Parameters

- **data** (*bytes-like object*) – Data to be decompressed.
- **max_length** (*int*) – Maximum size of returned data. When it's negative, the output size is unlimited. When it's non-negative, returns at most *max_length* bytes of decompressed data. If this limit is reached and further output can (or may) be produced, the *needs_input*

attribute will be set to `False`. In this case, the next call to this method may provide *data* as `b''` to obtain more of the output.

`needs_input`

If the *max_length* output limit in `decompress()` method has been reached, and the decompressor has (or may has) unconsumed input data, it will be set to `False`. In this case, pass `b''` to `decompress()` method may output further data.

If ignore this attribute when there is unconsumed input data, there will be a little performance loss because of extra memory copy. This flag can be True even all input data are consumed, when decompressor can be able to accept more data in some case.

`eof`

`True` means the end of the first frame has been reached. If decompress data after that, an `EOFError` exception will be raised. This flag can be False even all input data are consumed, when decompressor can be able to accept more data in some case.

`unused_data`

A bytes object. When PpmdDecompressor object stops after end mark, unused input data after the end mark. Otherwise this will be `b''`.

```
d1 = PpmdDecompressor()

decompressed_dat = d1.decompress(dat1)
decompressed_dat += d1.decompress(dat2)
decompressed_dat += d1.decompress(dat3)
```

CHAPTER

THREE

PPMD8 OBJECTS

Ppmd8Encoder and Ppmd8Decoder classes are intend to use general purpose text compression.

`class Ppmd8Encoder`

Encoder for PPMd Variant I version 2.

`__init__`(*max_order: int*, *mem_size: int*, *restore_method: int*)

The `max_order` parameter is between 2 to 64. `mem_size` is a memory size in bytes which the encoder use. `restore_method` should be either `PPMD8_RESTORE_METHOD_RESTART` or `PPMD8_RESTORE_METHOD_CUTOFF`.

`Ppmd8Encoder.encode`(*data: Union[bytes, bytearray, memoryview]*)

compress data, returning a bytes object containing copressed data. This data should be concatenated to the output produced by any preceding calls to the encode(). Some input may be kept in internal buffer for later processing.

`Ppmd8Encoder.flush`(*endmark: boolean*)

All pending input is processed, and bytes object containing the remaining compressed output is returned. After calling flush(), the encode() method cannot be called again; the only realistic action is to delete the object. flush() method releases some resource the object used.

When `endmark` is true (default), flush write endmark(-1) to end of archive, otherwise do not write anything and just flush.

`class Ppmd8Decoder`

Decoder for PPMd Variant I version 2.

`__init__`(*max_order: int*, *mem_size: int*, *restore_method*)

The `max_order` parameter is between 2 to 64. `mem_size` is a memory size in bytes which the encoder use.

These parameters should as same as one when encode the data.

`Ppmd8Decoder.decode`(*data: Union[bytes, bytearray, memoryview]*, *length: int*)

decode the given data and returns decoded data. When length is -1, maximum output data may be returned.

If decoder got the end mark, decode() method automatically flush all data and close some resource. When reached to end mark, `Ppmd8Decoder.eof` member become True.

When `Ppmd8Decoder.needs_input` is True, all input data is exhausted and need more input data to generate output. Otherwise, there are some data in internal buffer and reusable.

The decoder may return data which size is smaller than specified length, that is because size of input data is not enough to decode.

CHAPTER

FOUR

PPMD7 OBJECTS

Ppmd7Encoder and Ppmd7Decoder classes are designed to use as internal class for py7zr, python 7-zip compression/decompression library. Ppmd7Encoder and Ppmd7Decoder use a modified version of PPMd var.H that use the range coder from 7z.

`class` **`Ppmd7Encoder`**

Encoder for PPMd Variant H.

`__init__`(*max_order: int*, *mem_size: int*)

The `max_order` parameter is between 2 to 64. `mem_size` is a memory size in bytes which the encoder can use.

`Ppmd7Encoder.`**`encode`**(*data: Union[bytes, bytearray, memoryview]*)

Compress data, returning a bytes object containing compressed data for at least part of the data in data. This data should be concatenated to the output produced by any preceding calls to the encode() method. Some input may be kept in internal buffers for later processing.

`Ppmd7Encoder.`**`flush`**(*endmark: boolean*)

All pending input is processed, and bytes object containing the remaining compressed output is returned. After calling flush(), the encode() method cannot be called again; the only realistic action is to delete the object. When `endmark` is true, flush write endmark(-1) to end of archive, otherwise do not write (default).

`class` **`Ppmd7Decoder`**

Decoder for PPMd Variant H.

`__init__`(*max_order: int*, *mem_size: int*)

The `max_order` parameter is between 2 to 64. `mem_size` is a memory size in bytes which the encoder can use.

`Ppmd7Decoder.`**`decode`**(*data: Union[bytes, bytearray, memoryview]*, *length: int*)

returns decoded data that sizes is length.

decoder may return data which size is smaller than specified length, that is because size of input data is not enough to decode.

`Ppmd7Decoder.`**`flush`**(*length: int*)

All pending input is processed, and a bytes object containing the remaining uncompressed output of specified length is returned. After calling flush(), the decode() method cannot be called again; the only realistic action is to delete the object.

CHAPTER

FIVE

CONTRIBUTOR GUIDE

5.1 Development environment

If you're reading this, you're probably interested in contributing to pyppmd. Thank you very much! The purpose of this guide is to get you to the point where you can make improvements to the PyPPMd and share them with the rest of the team.

5.1.1 Setup Python and C compiler

The PyPPMd is written in the Python and C languages bound with both CFFI, C Foreign Function Interface, and CPython C/C++ API. CFFI is used for PyPy3 and CPython API is used for CPython.

Python installation for various platforms with various ways. You need to install Python environment which support *pip* command. Venv/Virtualenv is recommended for development.

We have a test suite with python 3.8 and pypy3. If you want to run all the test with these versions and variant on your local, you should install these versions. You can run test with CI environment on Github actions.

5.1.2 Get Early Feedback

If you are contributing, do not feel the need to sit on your contribution until it is perfectly polished and complete. It helps everyone involved for you to seek feedback as early as you possibly can. Submitting an early, unfinished version of your contribution for feedback in no way prejudices your chances of getting that contribution accepted, and can save you from putting a lot of work into a contribution that is not suitable for the project.

5.2 Code Contribution

5.2.1 Steps submitting code

When contributing code, you'll want to follow this checklist:

1. Fork the repository on GitHub.
2. Run the tox tests to confirm they all pass on your system. If they don't, you'll need to investigate why they fail. If you're unable to diagnose this yourself, raise it as a bug report.
3. Write tests that demonstrate your bug or feature. Ensure that they fail.
4. Make your change.
5. Run the entire test suite again using tox, confirming that all tests pass including the ones you just added.

6. Send a GitHub Pull Request to the main repository's master branch. GitHub Pull Requests are the expected method of code collaboration on this project.

5.2.2 Code review

Contribution will not be merged until they have been code reviewed. There are limited reviewer in the team, reviews from other contributors are also welcome. You should implemented a review feedback unless you strongly object to it.

5.2.3 Code style

The pyppmd uses the PEP8/Black code style. In addition to the standard PEP8, we have an extended guidelines.

- line length should not exceed 125 characters.
- Black format prettier is enforced.
- It also use MyPy static type check enforcement.

5.2.4 Test cases

There is three types of tests and we measures coverages;

- Unit tests for encode and decode, single data, and multiple data.
- Integration test with CSV file which size is larger than buffer size.
- Hypothesis fuzzing test.

All tests should be passed before merging.

5.3 C bindings development

Debuggng bindings has always been itchy task for developers. Even proprietary modern IDEs, such as PyCharm Professional/CLion, does not provide a cross debugging feature. Ref: https://youtrack.jetbrains.com/issue/CPP-5797

PyPpmd project source has a hacky way to do it.

5.3.1 CMake

The project has a *CMakeLists.txt* file to run cross-debugging, CMake is a cross-platform builder meta tool for C/C++ projects. You can run PyPpmd project as a C project using the file.

5.3.2 Jetbrains CLion

CLion is an IDE tool for C/C++ development that support CMake for build configuration. You can use CLion for PyPpmd development.

5.3.3 Dependency

- Python 3.8.x
- python development files (for example python3.8-dev package)
- venv
- GCC or CLang C/C++ compiler
- CMake 3.19 or later

When you want to change target python variation and version, please edit CMakeLists.txt#L8-L9

```
set(PY_VERSION 3.8)
set(Python_FIND_IMPLEMENTATIONS PyPy)
```

5.3.4 Manual build and run

TL;DR

```
mkdir cmake-build
cd cmake-build
cmake ..
make pytest_runner
gdb ./pytest_runner ../tests
```

pytest_runner is a generated program that help you run pytest under C/C++ debugger. You may want to run it on IDE environment.

You can also run pytest with tox

```
tox -e py38
```

5.3.5 Library build

```
cd cmake-build
make pyppmd
```

5.3.6 CMake targets and files

THere are several targets you can build.

pytest_runner:
A C++ program that launch python and pytest. The source code is generated by CMake configuration onto cmake build directory (cmake-build in above example).

generate_ext:
A virtual target to produce C extension for CPython.

pyppmd:
compile C files into static library file. Just convenient target for compilation.

venv.stamp:
interim target to produce virtualenv environment for pytest_runner

CHAPTER SIX

AUTHORS

`pyppmd` is written and maintained by Hiroshi Miura <miurahr@linux.com> Some part of `pyppmd` code are borrowed from `pyzstd` written by Ma Lin.

Contributors, listed alphabetically, are:

- **Sarthak Pati**
  macOS compilation
- **T Yamada**
  bug on race conditions on PPMd Variant I

CHAPTER

SEVEN

SECURITY POLICY

7.1 Supported Versions

Only recent version of pyppmd are currently being supported with security updates.

Version	Status
0.18.x	Development
< 0.18	not supported

7.2 Reporting a Vulnerability

Please disclose security vulnerabilities privately at miurahr@linux.com

CHAPTER

EIGHT

THIRD-PARTY SOFTWARE NOTICES AND INFORMATION

This project incorporates components derivered from the projects listed below. The original copyright notices and the licenses under which we received such components are set forth below.

1. lib/ppmd/* derived from 7-zip/p7Zip 16.00
2. lib/buffer/blockoutput.c derived from pyzstd v0.15.0

The other codes are original of PyPPMd project licensed under LGPLv2.1 or later.

8.1 7-zip/p7zip

C code under lib folder which originating from 7-zip are released under LGPL, and each sources are noted as follows.

- 2017-04-03 : Igor Pavlov : Public domain
- PPMd var.H (2001): Dmitry Shkarin : Public domain
- PPMd var.I (2002): Dmitry Shkarin : Public domain
- Carryless rangecoder (1999): Dmitry Subbotin : Public domain

8.2 pyzstd

A part of C extension code is a derived work of pyzstd which is licensed under BSD 3-Clause license.

CHAPTER

NINE

PYPPMD CHANGELOG

All notable changes to this project will be documented in this file.

9.1 Unreleased

9.2 v1.0.0

9.2.1 Changed

- Fix publish script to make sdist and upload it.
- Move CI on Azure pipelines
- Migrate forge site to CodeBerg.org
- Drop release-note and stale actions

9.3 v0.18.3

9.3.1 Added

- Release wheel for python 3.11 beta

9.3.2 Fixed

- CI: update setuptools before test run (#115)
- CI: fix error on tox test on aarch64.

9.3.3 Changed

- Bump pypa/cibuildwheel@2.7.0 (#116)
- Bump actions/setup-python@v4 (#114)
- Bump actions/download-artifact, actions/upload-artifact@v3 (#105,#106)
- CI: Test with python 3.11 beta(#112)
- Update license notifications
- Move C sources under `src/lib` folder

9.4 v0.18.2

9.4.1 Fixed

- Publish wheel package for python 3.10 on macos.
- pyproject.toml: add "version" as dynamic (#100)

9.4.2 Changed

- Update security policy to support version to be 0.18.x
- Move old changelog to Chanlog.old.rst

9.5 v0.18.1

9.5.1 Fixed

- Installation error with recent pip version (#94, #95) * Add metadata in pyproject.toml
- PPMd8: check double flush(#96)

9.6 v0.18.0

9.6.1 Fixed

- test: Fix fuzzer error with silent null byte (#89)
- test: 32bit test memory parameter too large(#90)
- PPMd7: avoid access violation on dealloc when failed in allocation (#91)
- PPMd7: decoder.eof and decoder.needs_input return proper value(#92)

9.6.2 Security

- PPMd7,PPMd8: fix struct definition by include process.h in windows This may cause crash on 32bit version of python on windows(#86)

9.6.3 Changed

- PPMd7: decompressor use threading(#85)

9.6.4 Added

- doc: Explanation of `Extra silent null byte` in README

9.7 v0.17.4

9.7.1 Fixed

- ppmd7: allow multiple decode without additional input data (#84)
- ppmd8: test: Fix fuzzer test program (#82)

9.7.2 Changed

- CI: bump actions/checkout@v3 (#81)
- CI: bump actions/setup-python@v3 (#80)
- CI: bump pypa/cibuildwheel@2.3.1 (#78)
- Update 32bit detection logic

9.8 v0.17.3

9.8.1 Fixed

- Build on MingW/MSYS2(#68,#69)

9.8.2 Added

- Test on Python 3.10.0, PyPY-3.6 and PyPy-3.7 (#71)

9.8.3 Changed

- CI: use pypa/ciwheelbuild(#70)
- CI: add dependabot(#70)
- Bump versions - CI: pypa/ciwheelbuild@2.2.2 - CI: run-on-arch@2.1.1 - CI: actions/stale@4
- CI: exclude pypy on windows
- CI: exclude cp310-macos because python 3.10 for macos is superceded
- CI: publish musllinux wheel
- CI: improve cibuildwheel performance

9.9 v0.17.1

9.9.1 Added

- Wheels for python 3.10

9.10 v0.17.0

9.10.1 Added

- unified API for variation H and I
- ppmd7, ppmd8: flag to control endmark(-1) addtions. defaults: ppmd7 without endmark, ppmd8: with endmark.

9.10.2 Changed

- Unified API to use Variant H, and Varant I version 2 from simple API. User can provide `variant` argument to the constractor. (#59)
- Allocate PPMD7Decompressor buffer variables from heap(#52)
- Replace pthread wrapper library to the verison of one made by Lockless. Inc. (#67)
- Refactoring internal variable namees, move thread shared variable into ThreadControl structure.

9.10.3 Fixed

- More robust PPMd8Decompressor by taking thread control variables and buffers from heap, and remove global variables.(#54)
- PPMD8Decoder: Deadlock on Windows(#67 and more)

9.10.4 Deprecated

9.10.5 Removed

- End-mark (0x01 0x00) mode(#62)

9.10.6 Security

9.11 v0.16.1

9.11.1 Added

- CI: add macOS as test matrix(#51)

9.11.2 Fixed

- Fix osX bulid error(#49,#50)

9.12 v0.16.0

9.12.1 Added

- PPMd8: support endmark option(#39)
- PPMd8: support restore_method option(#24, @cielavenir)
- Add pthread wrapper for macOS and Windows(#33)

9.12.2 Changed

- PPMd8: decompressor use threading(#24,#33)

9.12.3 Fixed

- PPMd8: Decompressor become wrong status when memory_size is smaller than file size(#24,#25,#28,#33,#45,#46)
- PPMd8: Decompressor allocate buffers by PyMem_Malloc() (#42)
- CMake: support CFFI extension generation(#30)
- CMake: support debug flag for extension development(#27)
- CMake: support pytest_runner on windows
- CI: run tox test on pull_request

9.13 v0.15.2

9.13.1 Added

- Add development note using cmake

9.13.2 Fixed

- Make CMake build script working

9.13.3 Security

- Hardening for multiplexing overflow(scan#1)

9.14 v0.15.1

9.14.1 Added

- Badge for conda-forge package(#19)

9.14.2 Changed

- Test document with tox

9.14.3 Fixed

- Fix setup.py: pyppmd.egg-info/SOURCES.txt not including full path
- Fix source package not include .git* files(#20)
- Fix compiler warning by cast.

9.15 v0.15.0

- Now development status is Beta.

9.15.1 Added

- Introduce PpmdCompressor and PpmdDecompressor class for stream compression.
- Introduce decompress_str() one-shot utility to return str object.

9.15.2 Changed

- decompress() always return bytes object.

9.15.3 Deprecated

- PPMd8: drop length mode for decompression and always use end mark mode.
- PPMd8: drop flush() method for decompression.

9.16 v0.14.0

9.16.1 Added

- Introduce compress() and decompress() one-shot utility - compress() accept bytes-like object or string. When string, encode it to UTF-8 first. - decompress() has an argument encoding, if specified, it returns string.
- C: CFFI: Introduce End-Mark mode for PPMd8

9.16.2 Changed

- C: Limit initial output buffer size as same as specified length.
- C: Allow python thread when decode/encode loop running.

9.17 v0.13.0

9.17.1 Added

- Benchmark test to show performance

9.17.2 Changed

- Change folder structures in source.
- Release resources on flush()

9.17.3 Fixed

- Fix input buffer overrun(#8)

9.18 v0.12.1

9.18.1 Fixed

- Fix dist of typing stubs

9.19 v0.12.0

9.19.1 Added

- add PPMd varietion I (PPMd8) - Ppmd8Encoder, Ppmd8Decoder class
- MyPy typing stubs

9.19.2 Changed

- switch to LGPLv2.1+ License
- Introduce flush() method for decode class.

9.19.3 Fixed

- Fix build error on Windows.

9.20 v0.11.1

9.20.1 Fixed

- Fix Packaging configuration

9.21 v0.11.0

9.21.1 Fixed

- Better error handling for memory management.

9.21.2 Changed

- Skip hypothesis tests on windows
- Limit hypothesis tests parameter under available memory.

9.22 v0.10.0

- First Alpha

CHAPTER TEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX